DAA PRACTICAL 6

Name – Avni Dongre

Section – A4

Roll No.- B3-46

Aim: Construction of OBST

Problem Statement: Smart Library Search Optimization
Task 1:
Scenario:
A university digital library system stores frequently accessed books using a binary search mechanism. The library admin wants to minimize the average search time for book lookups by arranging the book IDs optimally in a binary search tree.
Each book ID has a probability of being searched successfully and an associated probability for unsuccessful searches (when a book ID does not exist between two keys).
Your task is to determine the minimum expected cost of searching using an Optimal Binary Search Tree (OBST).

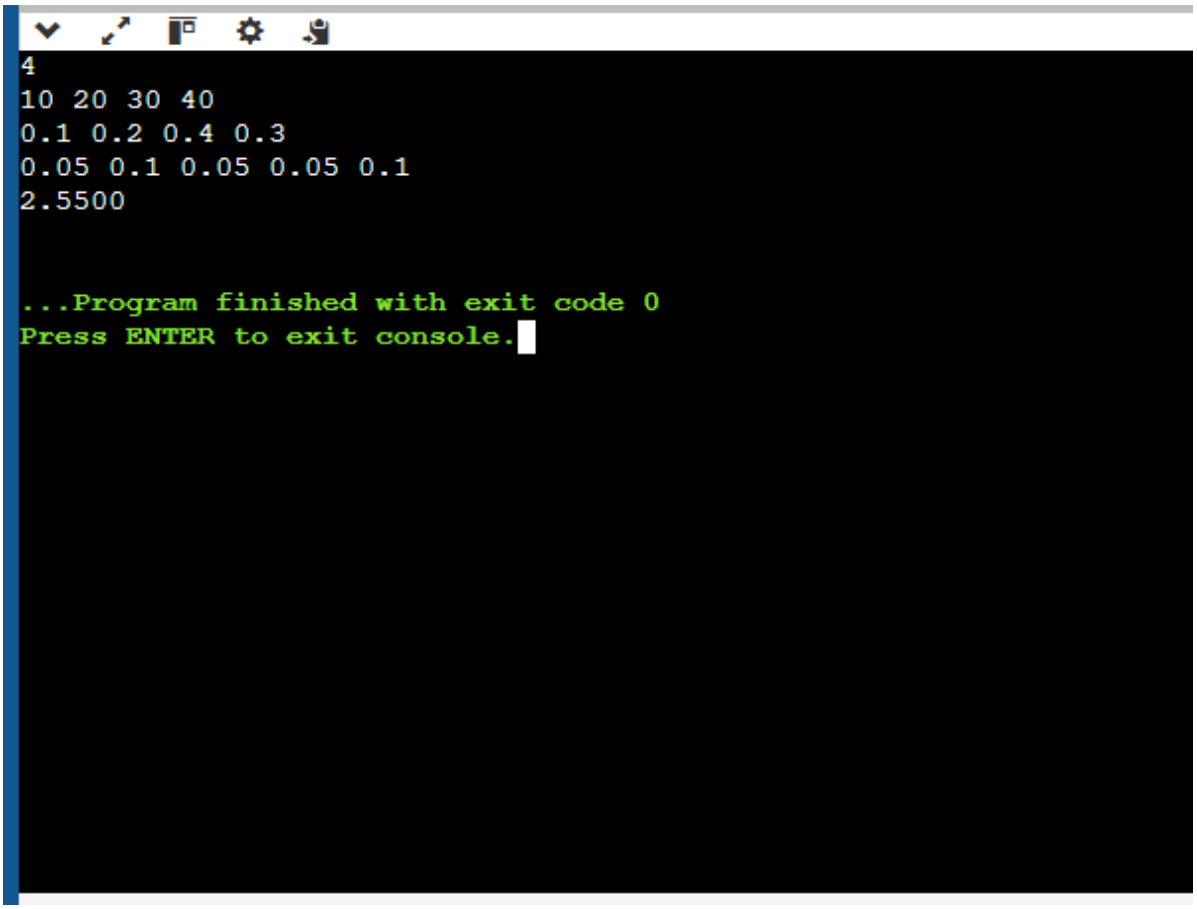
Code:-

```c
#include <stdio.h>
#include <math.h>

#define MAX_N 3

int main() {
    int i, j, k;
    int n;
    scanf("%d", &n);

    if (n > MAX_N) {
        printf("*****Entries exceed max limit*****\n");
        return 1;
    }

    int keys[MAX_N];
    double p[MAX_N];
    double q[MAX_N + 1];

    for (i = 0; i < n; i++) {
        scanf("%d", &keys[i]);
    }
```

```
for (i = 0; i < n; i++) {
    scanf("%lf", &p[i]);
}

for (i = 0; i <= n; i++) {
    scanf("%lf", &q[i]);
}

double cost[MAX_N][MAX_N];

for (i = 0; i < n; i++) {
    cost[i][i] = p[i] + q[i] + q[i + 1];
}

for (int length = 2; length <= n; length++) {
    for (i = 0; i <= n - length; i++) {
        j = i + length - 1;
        cost[i][j] = 1e9;

        double sum = 0;

        for (k = i; k <= j; k++) {
            sum += p[k];
        }

        for (k = i; k <= j + 1; k++) {
            sum += q[k];
        }

        for (k = i; k <= j; k++) {
            double left_cost = (k > i) ? cost[i][k - 1] : 0;
            double right_cost = (k < j) ? cost[k + 1][j] : 0;
            double current_cost = left_cost + right_cost + sum;

            if (current_cost < cost[i][j]) {
                cost[i][j] = current_cost;
            }
        }
    }
```

```
    }

    printf("%.4f\n", cost[0][n - 1]);

    return 0;
}
```

Output:-



Task 2 :

Code;

```
class Solution {
    // Function to calculate sum of frequencies from i to j
    int sum(int freq[], int i, int j) {
        int s = 0;
        for (int k = i; k <= j; k++) {
            s += freq[k];
        }
        return s;
    }

    // Function to return the minimum cost of the Optimal BST
    int optimalSearchTree(int keys[], int freq[], int n) {
        int[][] dp = new int[n][n];

        // Base case: Only one key
        for (int i = 0; i < n; i++) {
            dp[i][i] = freq[i];
        }

        // L is chain length
        for (int L = 2; L <= n; L++) {
            for (int i = 0; i <= n - L; i++) {
                int j = i + L - 1;
                dp[i][j] = Integer.MAX_VALUE;

                // Try making all keys in keys[i..j] as root
                for (int r = i; r <= j; r++) {
                    int cost = 0;

                    if (r > i) {
                        cost += dp[i][r - 1];
                    }
                    if (r < j) {
                        cost += dp[r + 1][j];
                    }

                    cost += sum(freq, i, j);

                    dp[i][j] = Math.min(dp[i][j], cost);
```

```
            }
        }
    }

    return dp[0][n - 1];
  }
}
```

Output :